CEWES MSRC/PET TR/98-42

MPI Inter-connection and Control

by

Graham E. Fagg
Kevin S. London

DoD HPC Modernization Program
Programming Environment and Training
CEWES MSRC

07h00998

Work funded wholly or in part by the DoD High Performance Computing Modernization Program CEWES Major Shared Resource Center through

Programming Environment and Training (PET)

Supported by Contract Number: DAHC 94-96-C0002
Nichols Research

MPI Inter-connection and Control

Graham E Fagg

Department of Computer Science

University of Tennessee

Knoxville

Tennessee 37996-1301, USA

Kevin S London

Department of Computer Science

University of Tennessee

Knoxville

Tennessee 37996-1301, USA

1.0 Introduction

This report details the use and installation of PVMPI 2.0 at the CEWES MSRC site. The first part of report details the PVMPI system and it's rational. The second section details specifics of this project to CEWES MSRC users and the future direction of this effort.

2.0 Technical Background

2.1 PVMPI Project Overview

Presently, different vendors' MPI implementations cannot inter-operate directly with each other. As a result, performance of distributed computing across different vendors' machines requires use of a single MPI implementation, such as MPICH. This solution may be sub-optimal since it cannot utilize the vendors' own optimized MPI implementations.

PVMPI, a software package currently under development at the University of Tennessee, provides the needed inter-operability between different vendors' optimized MPI implementations. As the name suggests **PVMPI** is a powerful combination of the proven and widely ported Parallel Virtual Machine (PVM) system and MPI. Two important features of

PVMPI are its transparent nature and its flexibility. **PVMPI** is transparent to MPI applications thus allowing intercommunication via all the MPI point-to-point calls. Additionally, **PVMPI** allows flexible control over MPI applications by providing access to all the process control and resource control functions available in the PVM virtual machine.

2.2 Message Passing Systems

The past several years have seen numerous efforts to address the deficiencies of the different message passing systems and to introduce a single standard for such systems. These efforts culminated in the first Message Passing Interface (MPI) standard, introduced in June 1994 [15].

Within a year, various implementations of MPI were available, including both commercial and public domain systems. One of MPI's prime goals was to produce a system that would allow manufacturers of high-performance massively parallel processing (MPP) computers to provide highly optimized and efficient implementations.

In contrast, systems such as PVM [1] were designed for clusters of computers, with the primary goals of portability, and ease-of-use. These have been achieved with little loss of performance [4] and with greater flexibility than the native communications system.

The aim of **PVMPI** was to interface the flexible process and virtual machine control from the PVM system with several optimized MPI communication systems thus allowing MPI applications the ability to inter-operate transparently across multiple heterogeneous hosts.

2.3 Virtual Machine Resource and Process Control

The PVM virtual machine is defined to be a dynamic collection of parallel and serial hosts. With the exception of one host in the PVM virtual machine, any number of hosts can join, leave, or fail without affecting the rest of the virtual machine. In addition, the PVM resource control API allows the user to add or delete hosts, check that a host is responding, shut down the virtual machine or be notified by a user-level message that a host has been added or deleted (intentionally or not).

The PVM virtual machine is very flexible in its process control capabilities. It can start serial, or parallel processes that may or may not be PVM applications. For example, PVM can spawn an MPI application as easily as it can spawn a PVM application. The PVM process control API allows any process to join or leave the virtual machine, start new processes by using a number of different selection criteria (including external schedulers, resource managers and/or taskers), signal or kill a process, test to check that a process is responding, and notify an arbitrary process if another disconnects from the PVM system.

In addition to the above virtual machine control functions, PVM provides plug-in interfaces for expanding its resource and process control capabilities. This extendibility has encouraged many projects to use PVM in different distributed computing environments such as Mist [14], dedicated schedulers [10], load balancers and process migration tools [5][16].

2.4 PVM Group Services

PVM provides the ability to group processes within the virtual machine. Groups are identified by a character string name. Processes can join and leave any number of groups at any time, thus making group membership completely dynamic. Processes are allocated instance numbers when they join a group, in the order of membership. The first join operation creates the group, and the group is destroyed when the membership falls to zero. Groups may have gaps in their membership as processes leave out of order. To improve performance, PVM allows group membership to be frozen by caching group details locally. Fully dynamic group caching is also available [11][12]. Many users only use the PVM group functions as a convenient naming/binding service.

2.5 MPI Communicators

Although the MPI standard does not specify how processes are started, it does dictate how MPI processes enroll into the MPI system. All MPI processes join the MPI system by calling MPI_Init and leave it by calling MPI_Finalize. Calling MPI_Init twice causes undefined behavior. Processes in MPI are arranged in rank order, from 0 to N-1, where N is the number of processes in a group. These process groups define the scope for all collective operations within that group. Communicators consist of a process group, context, topology information and local attribute caching. All MPI communications can only occur within a communicator.

Once all the expected MPI processes have started a common communicator is created by the system called MPI_COMM_WORLD. Communications between processes within the same communicator or group are referred to as *intra-communicator communications*. Communications between disjoint groups are *inter-communicator communications*. The formation of an inter-communicator requires two separate (non-overlapping) groups and a common communicator between the leaders of each group, as shown in Figure 1.

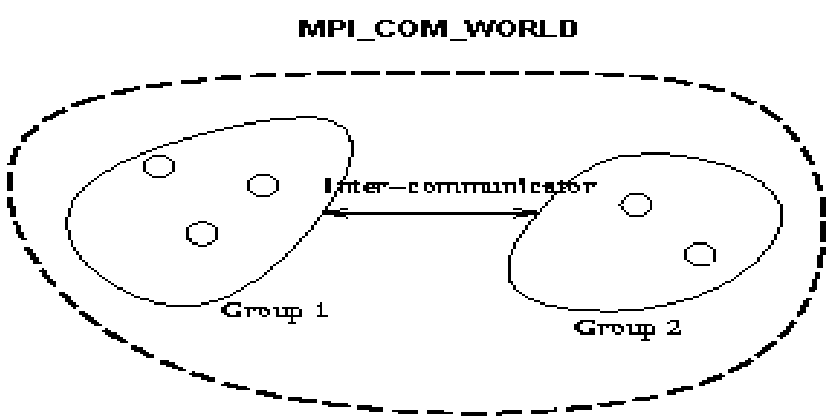

Figure 1. Inter-communicator formed inside a single MPI_COMM_WORLD

The MPI-1 standard does not provide a way to create an inter-communicator between two separately initiated MPI applications since no global communicator exists between them. The scope of each application is limited by its own MPI_COMM_WORLD, which by its nature is distinct from any other applications' MPI_COMM_WORLD. Since all internal details are hidden from the user and MPI communicators have relevance only within a particular run-time instance, MPI-1 implementations cannot inter-operate.

2.6 Related Work to PVMPI

Although several MPI implementations are built upon other established message-passing libraries such as Chameleon-based MPICH [7], LAM [3] and Unify [6], none allow true inter-operation between separate MPI applications across different MPI implementations.

LAM 6.X does allow some limited interaction between LAM only applications using a subset of functions from the dynamic process chapter of the proposed MPI-2 standards document.

Unify was originally proposed to *unify* or mate together the PVM and new MPI APIs. The intention was to enable users to take current PVM applications and slowly migrate toward complete MPI applications, without having to make the complete conceptual jump from one system to the other. The project never reached full maturity although it did address the difficulty of mapping identifiers between the PVM and MPI domains which it solved using additional function calls.

The only project known to the authors that attempts to directly interconnect MPI applications in a way similar to **PVMPI** is currently under way at the Computer Center of the University of Stuttgart in Germany (Rechenzentrum Universitaet Stuttgart). This project attempts to interconnect pairs of MPPs via specialist processes that use standard TCP/IP for communications. More on this project and the newly developed Nexus-MPI project [21] will be discussed in section 5.

3.0 PVMPI / MPI Connect project

3.1 The PVMPI System

We developed a prototype system **PVMPI2** [9] to study the issues of interconnecting MPI and PVM.

Four separate issues were addressed:

- mapping identifiers and managing MPI and PVM IDs
- transparent MPI message passing
- start-up facilities and process management
- performance effects

3.1.1 Mapping Identifiers

A tuple pair {process group, rank} or {communicator, rank} identifies a process in an MPI application. PVM provides similar functionality through use of the group library. The PVM tuple is {group name, instance}. **PVMPI** provides address mapping from the MPI tuple space to the PVM tuple space and vice versa. An initial prototype version of **PVMPI** [8] used such a system without any further translation (or hiding of mixed identifiers). The association of this tuple pair is achieved by registering each MPI process into a PVM group by a user level function call. A matching unassociate or leave call is also provided. The functions are available in both C and Fortran bindings:

info = PVMPI_Register(char *group, MPI_Comm comm, int *handle);

info = PVMPI_Leave(char *group);

call pvmpi_register(group, comm, handle, info)

call pvmpi_leave (group, info)

Both register and leave functions are collective and blocking: all processes in the specified MPI communicator have to participate. The PVMPI_Leave command is used to clean up MPI data structures and to leave the PVM system in an orderly way if required. Processes can register in multiple groups, although currently separate applications cannot register into a single group with this call (i.e. take the same named group). The register call takes each member of the communicator and makes it join a named PVM group so that its instance number within that group matches its MPI rank.

Since any two MPI applications may be executing on different systems using different implementations of MPI (or even different instances of the same version), the communicator usually has no meaning outside of any application callable library. The PVM group server, however, can be used to resolve identity when the group's names are unique. Once the application has registered, an external process can access it by using that process' group name and instance via the library calls **pvm_gettid()** and **pvm_getinst()**. When the groups have been fully formed, they are frozen and all their details are cached locally to reduce system overhead.

3.1.2 Transparent Messaging

The mixing of MPI and PVM group calls requires the understanding of two different message passing systems, their APIs, semantics and data formats. A better solution is to transparently provide inter-operability of MPI application by utilizing only the MPI API. As previously stated MPI uses communicators to identify message universes, and not PVM group names or TIDs. Thus the **PVMPI** could not allow users to utilize the original MPI calls for inter-application communication.

The solution was to allow the creation of virtual communicators that map either onto PVM and hence remote applications or onto real MPI intra-communicators for local communication. In order to provide transparency and handle all possible uses of communicators, all MPI routines using communicators were implemented using MPI’s profiling interface. This interface allows user library calls to be intercepted on a selective bases so that debugging and profiling tools can be linked into applications without any source code changes. Creating dual role communicators within MPI would require altering MPI's low level structure. As this was not feasible, an alternative approach was taken. **PVMPI** maintains its own concept of a communicator using a hash table to store the actual communication parameters. As communicators in MPI are opaque data structures this behavior has no impact on end user code. Thus **PVMPI** communicator usage is completely transparent as shown in Figure 2.

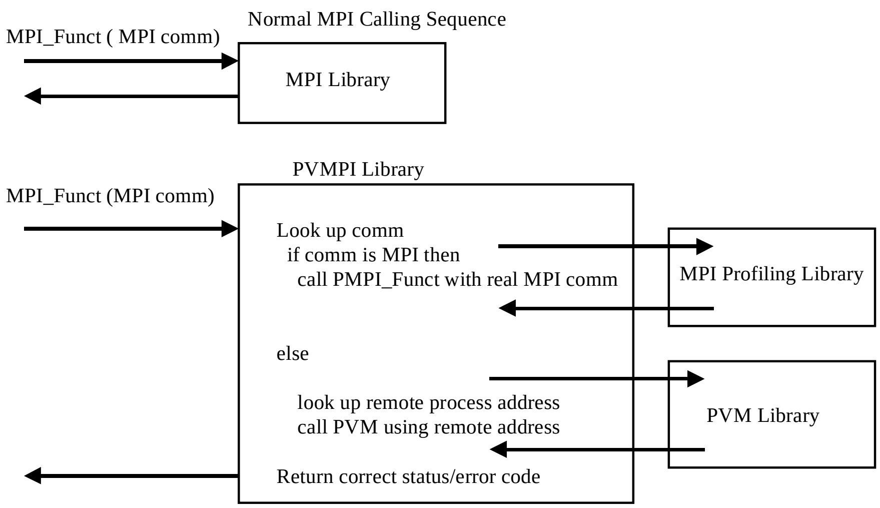

Figure 2. PVMPI usage of the MPI profiling interface to translate communicators.

Intra and inter communicator communications within a single application (MPI_COMM_WORLD) proceeds as normal, while inter-application communications proceed by the use of a **PVMPI** inter-communicator formed by using the PVMPI_Intercomm_create function:

```
info = PVMPI_Intercomm_create (int handle, char *gname, MPI_Comm *intercom);

call pvmpi_intercomm_create (handle, gname, intercom, info)
```

This function is almost identical to the normal MPI inter-communicator create call except that it takes a *handle* from the register function instead of a communicator to identify the local group, and a registered name for the remote group. The handle is used to differentiate between local groups registered under multiple names.

The default call is blocking and collective, although a non-blocking version has been implemented that can time-out or warn if the requested remote group has attempted to start and then failed, so that appropriate action can be taken to aid fault tolerance.

PVMPI inter-communicators are freed using the typical MPI function calls. They can be formed, destroyed and recreated without restriction. Once formed, they can be used exactly the same as a normal MPI inter-communicator except in the present version of **PVMPI** there is a restriction that they cannot be used in the formation of any new communicators. **PVMPI** inter-communicators allow the full range of point-to-point message passing calls inside MPI. Also supported are a number of data formatting and (un) packing options, including user derived data types (i.e. mixed striding and formats). Receive operations across inter-communicators *relies upon adequate buffering at the receiving end*, in-line with normal PVM operation.

3.1.3 Low-level Start-up Facilities

The spawning of MPI jobs from PVM requires different procedures depending upon the target system and the MPI implementation involved. The situation is complicated by the desire to avoid adding many additional spawn calls (the current intention of the MPI-2 forum). Instead, a number of different MPI implementation specific taskers have been developed that intercept the internal PVM spawn messages and then correctly initiate the MPI applications as required.

3.1.4 Process Management under a General Resource Manager

The PVM GRM [10] can be used with specialized **PVMPI** taskers to manage MPI applications in an efficient and simple manner. This provides improved performance [13] and better flexibility than that of a simple host file utilized by most MPIRUN systems. When a user's spawn request is issued it is intercepted by the GRM and an attempt is made to optimize the placement of tasks upon available hosts. If the placement is specialized then appropriate taskers are used. Figure 3 shows a system with three clusters of machines: one each for MPICH, LAM and general-purpose jobs. In this figure the start request causes two MPICH nodes to be selected by the GRM, then the MPICH tasker starts the actual processes.

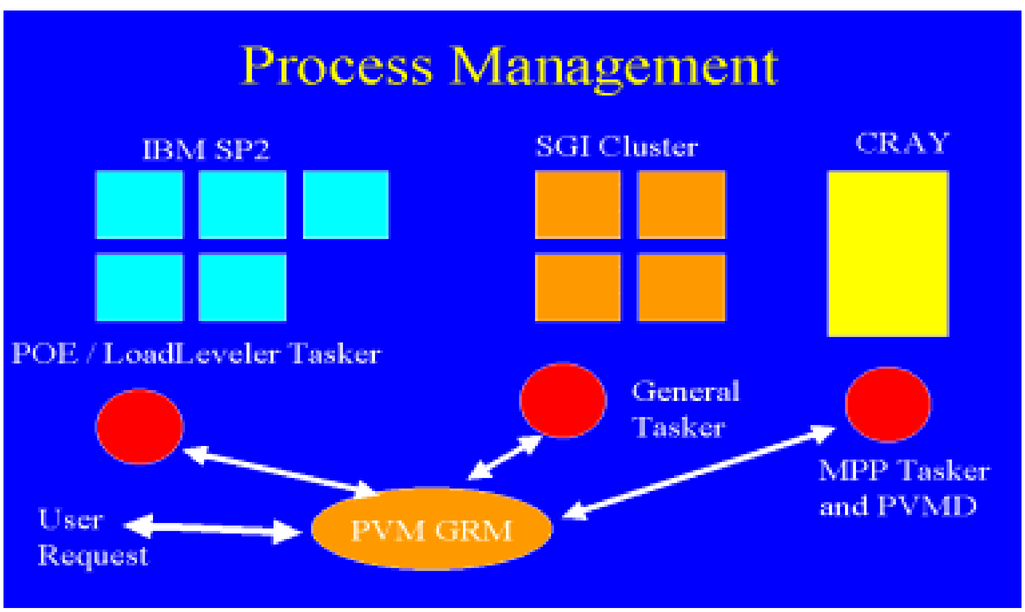

Figure 3. GRM and Tasker interaction.

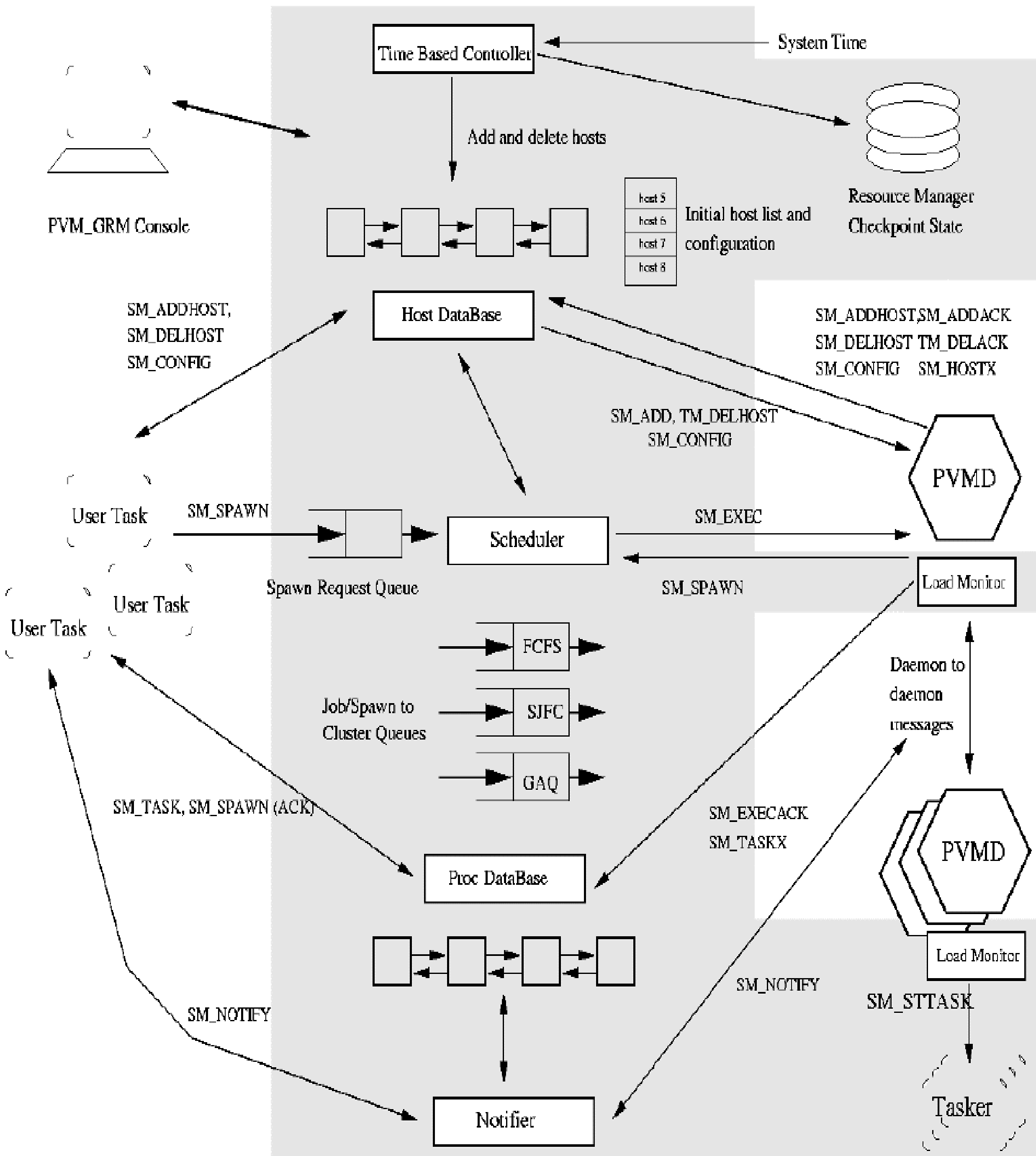

Figure 4 Internal details of the PVM GRM system.

3.1.5 Performance Effects

The performance effects of **PVMPI** were measured in two different areas:

- single MPI implementation instances / Intra-communication
- across MPI implementations instances / Inter-communication

Intra-communication performance was taken as the performance (or performance loss to/) of a normal MPI application that did not attempt any external communication outside of any particular MPI implementation instance, i.e., a stand-alone application running purely under just LAM, MPICH or IBM's native MPI etc.

This was found to un-measurable accurately on a workstation cluster connected with 100Mb/s Ethernet. Specific tests on CEWES MSRC MPPs are on going, although for operations on the SGI O2000s a slight performance loss was found for small messages as expected. This is shown in Figure 5 below.

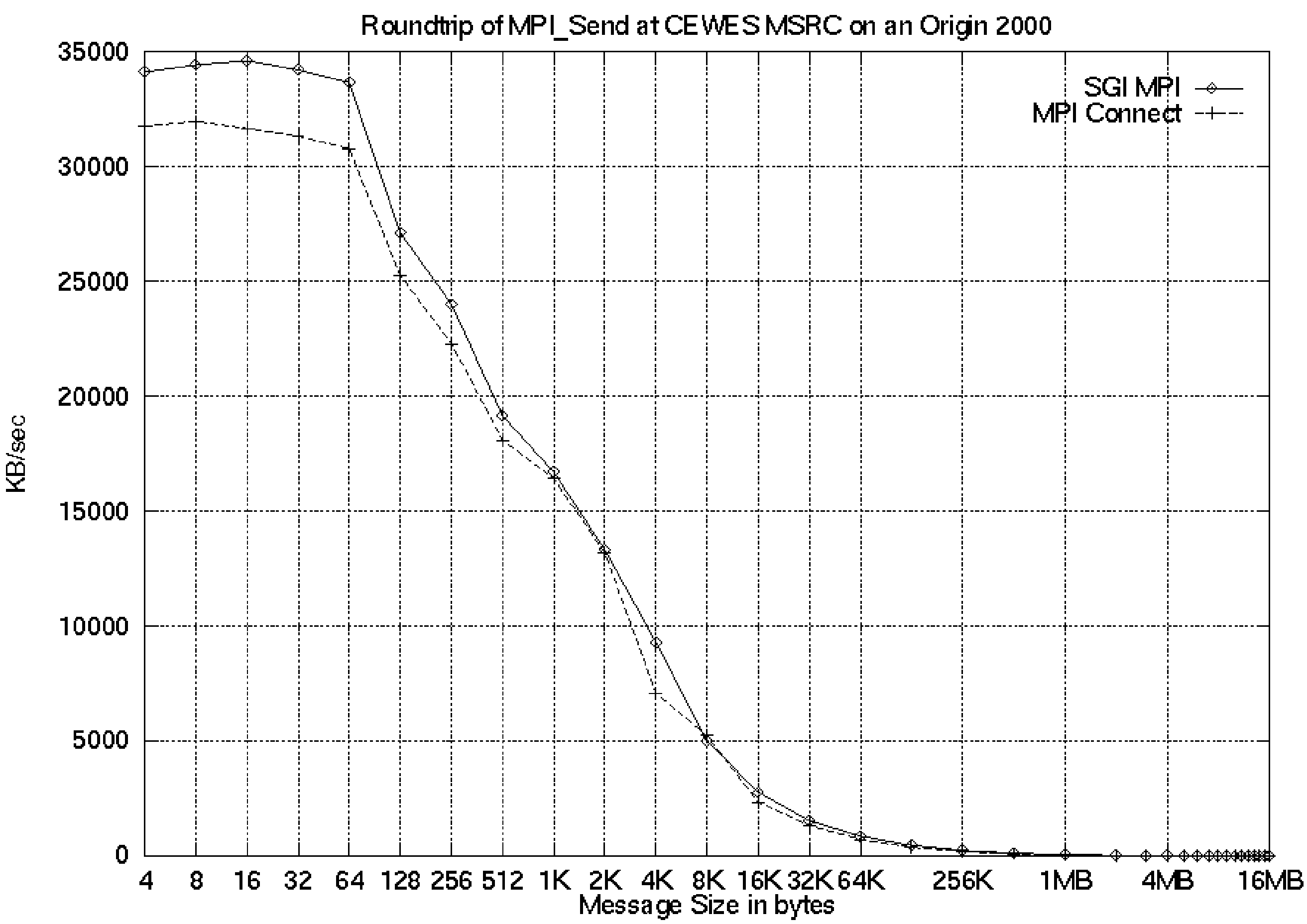

Figure 5. Bandwidth calculated from the roundtrip time between two processors on separate nodes of the CEWES MSRC O2000 machine.

Slight overhead from looking up communicators and passing the calls to the MPI profiling interface is shown in Figure 5 for small messages (typical < 1KB). The benchmark software used was MPBENCH from the University of Tennessee, which was developed as part of the Performance Evaluation Program.

Inter-communication performance is considered communication between different MPI implementation instances, whether running upon different hardware systems or just independently started implementations upon the same platform.

Initial tests have shown typical PVM communication speeds (latency and bandwidth) depending upon the inter-connection media and type of communication.

4.0 Current Work

4.1 Running PVMPI under CEWES MSRC systems

This has been attempted in several stages. Initially the **PVMPI** system used an older version of PVM (3.3.8), which had been modified to allow enhanced control over the PVM taskers than was provided with the default public domain version. Since then, the **PVMPI** system has been upgraded to use PVM 3.4, which provides better naming services and user definable message communication contexts. PVM 3.4 has been installed with a base version of **PVMPI** on the CEWES MSRC SGI PCAs, O2000s and SP systems.

4.1.1 SGI O2000 and PCAs

PVMPI was ported successfully to the SGI version of MPI on the PCA and O2000 systems. Only minor changes were necessary to allow **PVMPI** to correctly interact with SGI's version of MPI. These changes coupled with the ease of running a PVMD on the same machine as an MPI application have produced the simplest port.

4.1.2 IBM SP

PVMPI was ported and installed on the SP system, which presented a few conversion problems initially. This port has now been updated and is running correctly.

Two separate restrictions were found:

(1) The version of PVM used on the SP nodes is the workstation version configured to use the high speed switch (when available), and not the SP2MPI version. Under the normal partition management software, in this case IBM's POE, PVM daemons could only be started on the interactive nodes. (This was also found to be a problem at other DoD MSRC sites).

(2) In the case of CEWES MSRC, the IBM POE system has to be invoked via the PBS job control system (PBS-POE). This has the problem that after setting up the PVM Virtual Machine (running a daemon on each node where the MPI applications will execute), the PBS system kills all other user processes when starting the MPI application, i.e., removing the underlying PVM virtual machine and thus disallowing the MPI application from inter-operating with other machines. Currently we can only test one node at a time, which is

sufficient for testing inter-machine communication at CEWES MSRC, with full SP testing occurring at the Aeronautical Systems Center MSRC.

Other than changing the runtime system, the solution is to remove the restriction of having every MPI node have its own PVM daemon available. In this case, native messaging is used to pass external messages to an additional MPI process that acts as an external gateway. This is explained further in section 5.

4.1.3 Cray T3E

The Cray T3E presents a different problem. Although there is a version of PVM available for the T3E, it is the VENDOR version and does not inter-operate with the public domain version of PVM from ORNL.

There is however a solution in that it's possible to utilize TCP sockets directly on any node and then send/receive messages by utilizing these sockets. To reduce complexity, normal MPI communications can efficiently pass external messages to additional MPI processes that handle these external messages via their own TCP stacks. This is the solution being used by a package called PACX from the Stuttgart Computer Center in Germany that allows only T3Es to inter-operate. A similar solution is being developed that also solves the IBM SP problems indicated above.

Currently, no interoperation is possible between the T3E and different vendor machines until the listed modifications to **PVMPI** have been completed.

4.2 Runtime systems / Job Control / Q-Sub / PBS-POE

The runtime systems have difficulties in allowing both PVM and MPI to co-exist in some systems, even when appropriate changes have been made to the GRM system to allow a single point for job control. As the new version of **PVMPI** comes on line, these problems will diminish.

4.3 Other Support Systems

4.3.1 PVM 3.4b

The latest version of PVM, PVM 3.4b, was installed on all applicable hardware at CEWES MSRC (IBM SP and SGI PCA/O2000). This version of PVM has internal message contexts, which improves PVM's ability to handle multiple communications in a safer manner. This has allowed the use of **PVMPI** with simpler message routing/handling overheads since it does not have to partly unpack messages to determine routing requirements.

5.0 Future Work

5.1 New Architecture Support

Supporting the current IBM SP systems requires a change in structure of the **PVMPI** system, which can be combined with the changes required to support **PVMPI** for other architectures such as the Cray T3E and the Intel Paragon XP series machines.

The changes basically require the routing and encoding used by external (in) bound messages to use additional processes that have external access, instead of assuming all nodes can access PVM which provides all external access. All internal messaging to these nodes will use the vendor optimized MPI communications. In the case of the Cray T3E and Intel Paragon XP machines, these communication nodes will not have access to PVM and thus will be forced to utilize a different inter-machine communication medium such as SNIPE [19]. SNIPE uses either TCP/IP or the NEXUS [21] communications library from Globus [20]. SNIPE provides its name resolution service via the Resource Catalogue and Distribution Service (RCDS) [22], which replaces PVM's group server *PVMGS*. Figures 6a-6d show the different communication layouts and process placements involved for each of the different architectures.

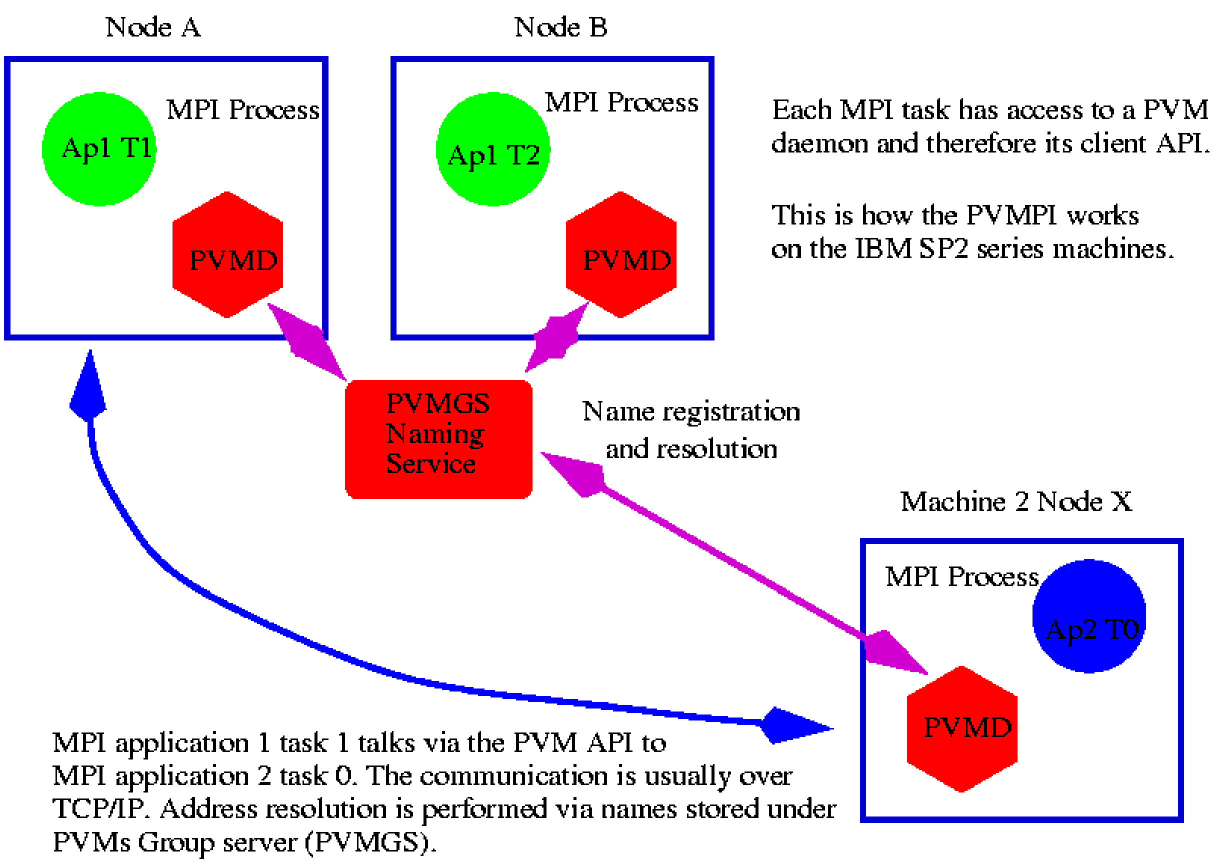

Figure 6a. All MPI processes have access to their own PVMD.

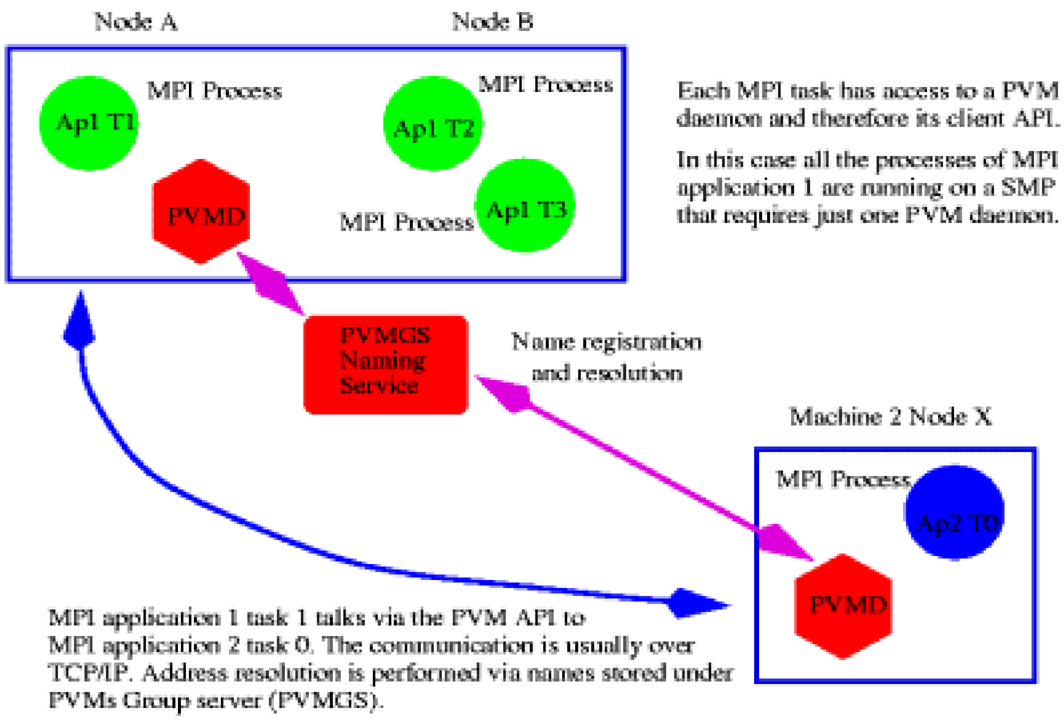

Figure 6b. MPI processes have to share a PVM daemon but can access the PVM API directly.

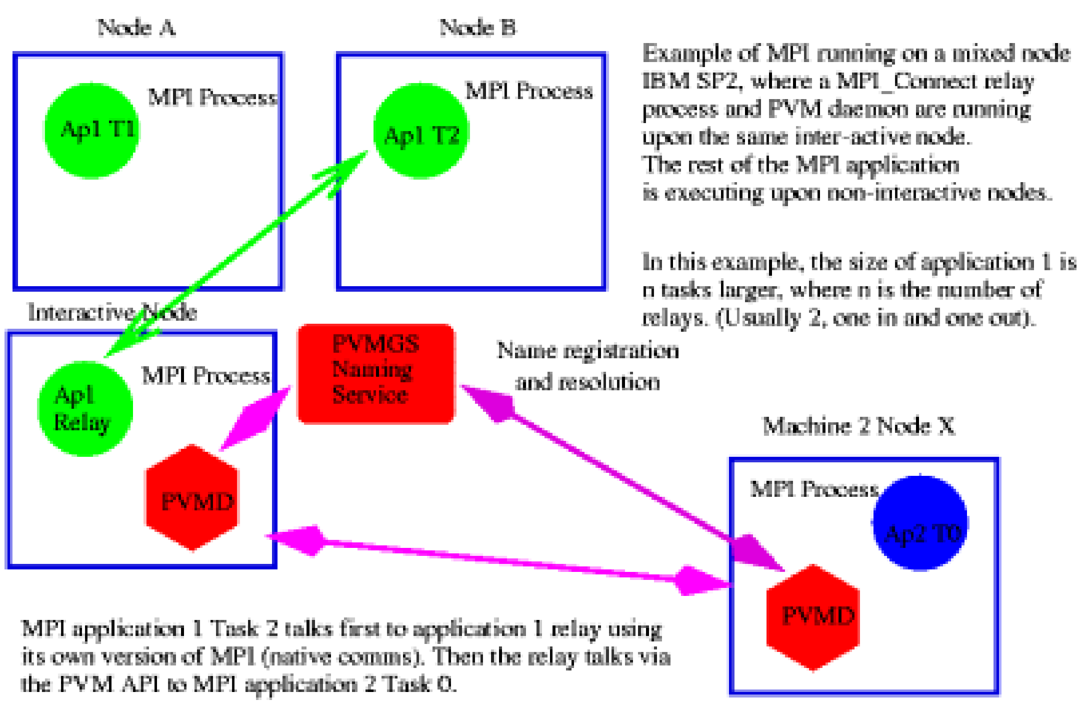

Figure 6c. Only a few MPI processes can access a PVM daemon directly. Thus external communication must be relayed to these processes for additional routing.

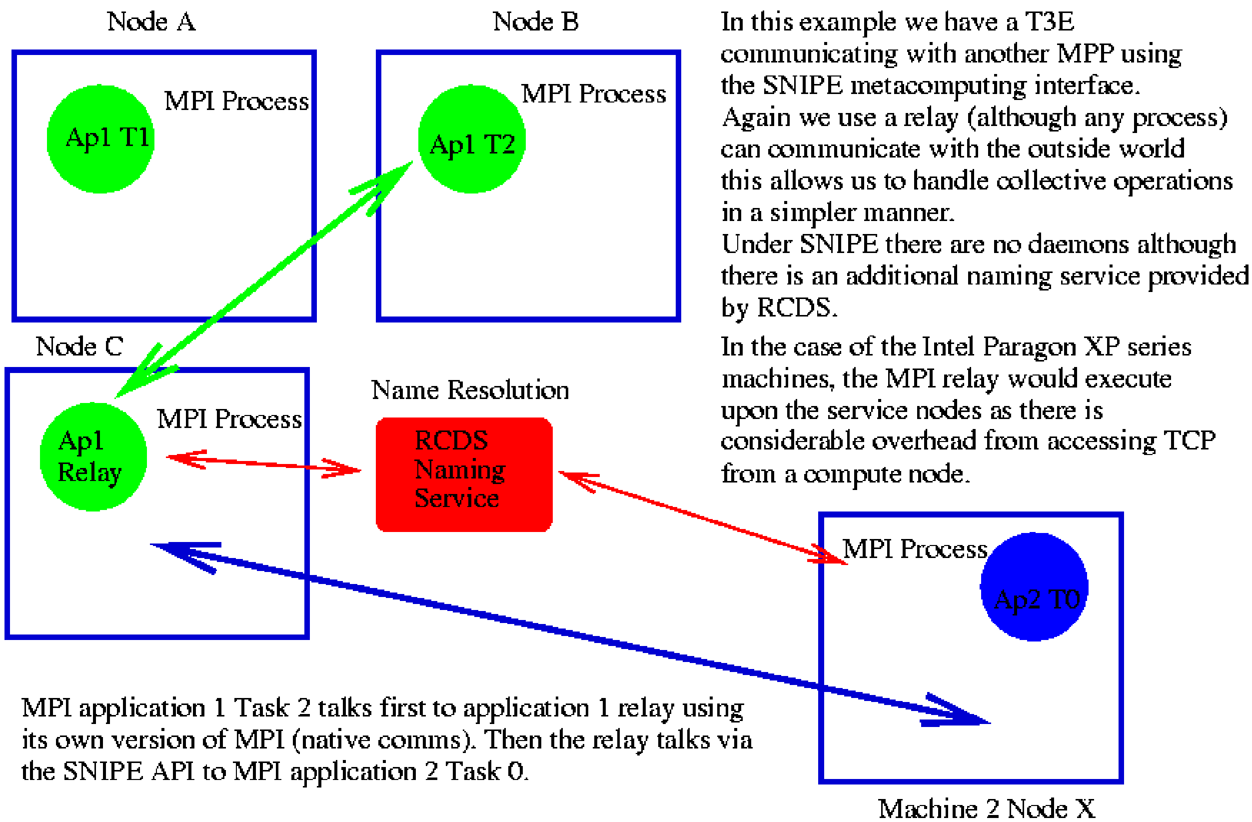

Figure 6d. Interconnecting MPPs using SNIPE instead of PVM to avoid possible problems maintaining PVM daemon processes.

5.2 Different Programming Models

The current programming model is based upon the idea of two or more distinct applications inter-operating via the normal MPI API. Their organization may be of a traditional peer-to-peer or client-server model. The **PVMPI** system itself does not inflict any addition control over the applications and it is entirely up to the application developer to add additional control structures to alter the programming model.

With the restructuring of the code, it is possible through the capturing of all the MPI API to allow the two separate applications to interact as if they were a single application. For example, the programmer would no longer need to register applications under names and then create communicators between them. For some applications this would be an advantage and a more natural method of structuring.

The only disadvantage for the system's side of the software is that all the collective and topology functions in MPI would now have to be supported. This has been a hindrance in other projects similar to **PVMPI** such as PACX [2][23], which up to now only support a limited set of functions that are required by the specific applications it was developed to support.

5.3 Enhancements

One of **PVMPI/MPI_Connects** advantages is that it fully supports user derived data types. This can be extended with some of the MPI-2 parallel IO features to allow two separate applications to exchange and archive data in an efficient and portable manner. Currently the MPI-2 specification does not indicate how different MPI implementations would create joint files when using the parallel interface. We hope to provide some support for this so that when a single application is executed across multiple platforms, the parallel IO continues seamlessly.

Using relay processes also provides a useful single point for all external communications and thus allows:

- Encryption of messages. Useful between compute servers running at MSRC sites talking to clients at off-site.
- Compression of communication. When passing large data messages over low-bandwidth connections this can reduce the period of synchronous/collective operations.

5.4 User Applications

As part of our program to support CEWES MSRC users we aim to assist the year three focused effort on "Climate, Weather, Ocean Modeling Simulation" in association with Ohio State University. This project aims at parallelizing and coupling the WAM wave model with the CH3D circulation model. In its later stages, the NCAR MM5 atmospheric model and COSED sediment models will also be integrated. The project is an ideal candidate as all four models have slightly different computational and communication requirements and therefore are suited to different hardware platforms. Currently most models are coupled by sharing data files on shared file systems. Prof. Keith Bedford is leading this project from Ohio State University.

5.5 Across MSRC Site Operation

With the transfer from PVM to non-virtual machine based communications and name resolution, it becomes more feasible to extend operation across MSRC sites. Thus, users may start an application that runs in part, on different machines at each of the MSRC sites, allowing a much larger problem size to be solved.

6.0 Conclusions

The PVMPI system solves the lack of interoperability between MPI-1 implementations. It allows the user to run applications across different hardware systems while still utilizing the vendors' optimized MPI implementations on each system. PVMPI usage is transparent to the end user and its usage requires only three additional calls (`PVMPI_register, PVMPI_leave` and `PVMPI_Intercom_create`). Additionally, it provides flexible process management and assists in efficient use of networked resources.

7.0 References

[1] A. L. Beguelin, J. J. Dongarra, A. Geist, R. J. Manchek, and V. S. Sunderam. Heterogeneous Network Computing. *Sixth SIAM Conference on Parallel Processing*, 1993.

[2] Thomas Beisel. ``Ein effizientes Message-Passing-Interface (MPI) fuer HiPPI'', *Diploma thesis*, University of Stuttgart, 1996.

[3] Greg Burns, Raja Daoud and James Vaigl. LAM, An Open Cluster Environment for MPI. Technical report, Ohio Supercomputer Center, Columbus, Ohio, 1994.

[4] Henri Casanova, Jack Dongarra and Weicheng Jiang. The Performance of PVM on MPP Systems. *Department of Computer Science Technical Report CS-95-301*. University of Tennessee at Knoxville, Knoxville, TN. August 1995.

[5] J. Casas, R. Konuru, S. Otto, R. Prouty, and J. Walpole. Adaptive Load Migration Systems for PVM. *Supercomputing'94 Proceedings* , pp. 390-399, IEEE Computer Society Press, 1994.

[6] Fei-Chen Cheng. Unifying the MPI and PVM 3 Systems. Technical report, Department of Computer Science, Mississippi State University, May 1994.

[7] Nathan Doss, William Gropp, Ewing Lusk and Anthony Skjellum. A model implementation of MPI. *Technical report MCS-P393-1193*, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL 60439, 1993.

[8] Graham E. Fagg and Jack J. Dongarra, PVMPI: An Integration of the PVM and MPI Systems. *Calculateours Paralle`les*, Paris, Vol 8/2, pp. 151-166, June 1996.

[9] Graham E. Fagg, Jack J. Dongarra and Al Geist, PVMPI provides Interoperability between MPI Implementations, *Proceedings of Eight SIAM conference on Parallel Processing*, March 1997.

[10] Graham E. Fagg, Kevin London and Jack J. Dongarra, Taskers and General Resource Manager: PVM supporting DCE Process Management, *Proceeding of the third EuroPVM group meeting*, Munich, Springer Verlag, October 1996.

[11] G.E. Fagg, R.J. Loader, P.R. Minchinton and S.A. Williams. Improved Group Services for PVM. *Proceeding of 1995 PVM Users Group Meeting*, Pittsburgh, pp.6, May 1995.

[12] Graham E. Fagg, Roger J. Loader and Shirley A. Williams. Compiling for Groups. *Proceeding of EuroPVM 9*5, pp. 77-82, Hermes, Paris, 1995.

[13] Graham E. Fagg and Shirley A. Williams. Improved Program Performance using a cluster of Workstations. *Parallel Algorithms and Applications*, Vol 7, pp. 233-236, 1995.

[14] R. Konuru, J. Casas, S. Otto, R. Prouty and J. Walpole. A User-Level Process Package for PVM. Scalable High Performance Computing Conference, pp. 48-55, *IEEE Computer Society Press*, 1994.

[15] Message Passing Interface Forum. MPI: A Message-Passing Interface Standard. *International Journal of Supercomputer Applications*, 8(3/4), 1994. Special issue on MPI.

[16] Georg Stellner and Jim Pruyne. Resource Management and Check-pointing for PVM, *Proceeding of EuroPVM 95*, pp. 130-136, Hermes, Paris, 1995.

[17] Louis H. Turcotte. ``A Survey of Software Environments for Exploiting Networked Computing Resources", *MSSU-EIRS-ERC-93-2*, Engineering Research Center for Computational Field Simulation, Mississippi State University, February 1993.

[18] Shirley A. Williams and Graham E. Fagg. "A Comparison of Developing Codes for Distributed and Parallel Architectures", *Proc of BCS PPSG UK Parallel 96*, pp. 110-118, Springer Verlag, London, 1996.

[19] Graham E Fagg, Keith Moore, Jack Dongarra and Al Geist. "Scalable Networked Information Processing Environment (SNIPE)", *Proc of SuperComputing 97*, San Jose, November 1997.

[20] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *International Journal of Supercomputer Applications*, (to appear) 1997. Available at ftp://ftp.mcs.anl.gov/pub/nexus/reports/globus.ps.Z

[21] Ian Foster, Carl Kesselman, Robert Olson, and Steve Tuecke. Nexus: An interoperability toolkit for parallel and distributed computer systems. *Technical Report ANL/MCS-TM-189*, Argonne National Laboratory, 1994.

[22] K. Moore, S. Browne, J. Cox, and J. Gettler. "The Resource Cataloging and Distribution System", Technical report, Computer Science Dept., University of Tennessee, December 1996.

[23] Thomas Beisel, Edgar Gabriel and Michael Resch, "An Extension to MPI for Distributed Computing on MPPs", *Proc. of the 4th EuroPVM/MPI User Group Meeting*, Cracow, Poland. Published in Recent Advances in PVM and MPI, Lecture Notes in Computer Science Vol. 1332, Springer, Nov97, pp75-82.